Science United: Implementation

David P. Anderson
Spaces Sciences Laboratory
University of California, Berkeley

April 27, 2018

1. Overview

Science United (SU) is a coordinator for BOINC-based volunteer computing.

1.1 Development and administration

SU is being designed and developed by my group, with funding from the NSF.

SU's software is open source, distributed under the LGPLv3 license. It's available on Github.

SU currently is hosted on servers at UC Berkeley.

2. Keywords

As a basis for volunteer preferences, we have defined a system of "keywords" for describing jobs. The system has the following structure:

- Category: currently "science area" or "location". We could also consider categories for openness of results, but this may be hard to define.
- Hierarchy: each level N+1 keyword is a child of a single level N keyword. The hierarchy can change over time.
- Each keyword has a permanent integer ID, and short and long textual descriptions that can change over time.

2.1 Project and job attributes

Each job has an associated set of keywords. These are specified in the job submission process. For example, if a TACC job is submitted by a cancer researcher at UCB, the attributes would include "cancer research" and "UCB".

Each project publishes a set of keywords, and for each keyword the estimated fraction of current jobs having that keyword. If the fraction is one, the keyword is implicitly associated with all the project's jobs. The set of project keywords can change over time, reflecting changes in the project's workload. SU polls projects for their keywords.

2.2 Keyword preferences

A volunteer can specify, on the SU web site, a set of "preferences", represented as a map from keywords to (yes, no, maybe). "No" means don't send jobs with that keyword. "Yes" means preferentially send jobs with that keyword. A "no" for a keyword trumps "yes" for descendant keywords.

If a project has a keyword with job fraction 1, and a volunteer has "no" for that keyword, the volunteer's devices should not be attached to that project.

When a new keyword is added, the default setting is "maybe" for all volunteers. Volunteers are notified of the new keyword in case they want to change this.

2.3 Keyword-based scheduling

The BOINC job dispatcher uses "score-based" scheduling: in deciding which jobs to send to a device, the scheduler computes a score for each job that includes a number of different factors; it then sends the highest-scoring jobs. We have extended this to include keywords. For each of the job's keywords, if the volunteer has "yes" the score is incremented, and if "no" the score is zero (meaning don't send the job).

This lets volunteer preferences be enforced for projects with applications in a range of science areas, and/or running jobs are a range of client institutions.

3. Resource usage accounting

SU does accounting of resource usage. This serves two main purposes:

- It provides basis for volunteer incentives: for example, graphs showing work done recently, work "milestones", and so on.
- It provides a basis for resource allocation to projects, and gives an estimate of the system-wide throughput.

BOINC has a sophisticated credit system for estimating the FLOPs performed by a completed job. It is fairly "cheat-proof": it is difficult to get credit for computation not actually performed. However, this is based in part on job replication, meaning that credit for a job may not be granted until the companion job is completed, which could take weeks. This makes it undesirable for volunteer incentives.

Instead, SU uses a quantity called "estimated credit" (EC), which is maintained by the BOINC client on a per-job and per-project basis, based on the runtime of jobs and the peak FLOPS of the processors they use. EC is a cruder estimate than credit, and it is not cheat-proof. But it accumulates continuously, without waiting for job completion or validation.

4. The resource share model

SU's resource allocation mechanism (see below) incorporates several factors. One of these is the notion of "resource share": how much resources a projects gets relative to other projects. We use the following model:

- Each project P has a "resource share" R_P. The fraction of resources available to P (over a sufficiently long period, and all other things, such as keywords, being equal) is at least R_P/R_{total}, where R_{total} is the sum of R_P over all projects. Resource shares may change over time.
- To enforce resource shares, each project P has a "balance" B_P which represents how many FLOPs are "owed" to P. During a given accounting period (say, a day) B_P is incremented by the number of FLOPs actually performed during the period, times R_P/R_{total} . The balance is capped at a limit L_P. Balance is in units of floating-point operation (FLOPs).
- When computation is reported by a client for a project P, B_P is decremented by the amount of computation.
- At any point, computing resources are preferentially assigned to projects P for which B_P is greatest.

This model handles both continuous and sporadic workloads well. For a project P with continuous workload, B_P will remain around zero. For a project P with sporadic workload, B_P will usually be at the limit L_P. Then P generates a burst of work, it will have priority over the continuous-workload projects, and will get done quickly.

When a computer is assigned to a project, there will be a delay of about a day (the client polling period) until computation is reported by the client to SU. This means that the same project (the one for which B_P is greatest) will be assigned to all hosts during that period. This is undesirable. To solve this problem, we maintain a separate "projected balance" for each project. This is decremented by the appropriate amount when a computer is assigned to the project. At the end of each accounting period, projected balances are reset to the balances.

4.1 Allocation

Initially, all projects will have equal shares (R_P values). However, at some point we'll want the capability of giving projects greater shares, on either a temporary or permanent basis. These "allocations" will be consist of elevating R_P to a given, starting and ending at given times.

Allocation decisions will be made by the SU committee, according to a published policy. Allocations could be based on merit, on special requirements, or on payment.

4.2 Performance guarantees

Some existing HTC systems can offer "performance guarantees": a client can be guaranteed a given throughput over a given period of time with very high probability. It's desirable to offer analogous guarantees with VC resources.

The performance of a pool of volunteer computers can vary, in terms of both throughput and job latency. However, with a large pool, these quantities change slowly over time, and we can measure

these changes and establish the statistics of their change. For example, given the total throughput T at a given time, we could find a $T_0 < T$ such that total throughput will remain above T_0 for a week.

In this way, it's likely that the "allocations" described above can be mapped to specific performance guarantees; the details remain to be figured out.

Such guarantees would be project-level. Can we provide performance guarantees to a particular job submitter within a project that serves lots of job submitters? This is more complex but it may be possible. The BOINC server software uses the linear-bounded model, exactly as described above, to allocate resources among computing job submitters within the project. It's possible that the combination of a project-level allocation and a submitter-level allocation provide some form of performance guarantee to the submitter.

5. Resource allocation

The central function of SU is to allocate resources among projects. It does this by dynamically assigning projects to computers.

The resource allocation policy has several possibly conflicting goals:

- It must enforce volunteer preferences: if 100% of volunteers said "no" to cancer research, SU can't assign computers to a project that does only cancer research. However, we anticipate that volunteer preferences will be primarily positive ("yes"). This gives SU some freedom, since "yes" is non-binding: SU is free to assign "maybe" work even if "yes" work is available.
- It tries to divide resources among projects based on the "resource share" model described above.
- It tries to maximize total throughput. For example, if a host has a GPU, it should be assigned at least one project that can use the GPU.

Clients using SU periodically (perhaps once per day) issue an AM RPC. The request message includes a list of currently-attached projects and their work totals; these are used to update account records. The reply message includes a list of projects to attach to. The client detaches from any projects not on this list. For each project, the reply specifies a "resource share": a value of zero means that the client should do work for this project only if none of the other projects have work available.

Being attached to a project has a disk overhead; the client caches applications files for the project, which may include large VM image files. Hence we want to limit the number of projects each client is attached to. On the other hand, if a project has a large disk footprint on a client, we may want the client to remain attached (with a zero resource share) even if we don't want it to compute at this point.

6. The project assignment algorithm

With these factors in mind, here is a sketch of the project assignment algorithm currently used by SU:

- For each project (of all the projects managed by SU) compute a score for this client. This score includes several components:
    - Don't use the project if it doesn't support the host's platform.
    - A keyword factor. Increment the score if the project has keywords in the volunteer's "yes" list. Don't use the project if it has a "no" keyword with job fraction one.
    - Increment the score if the host has VirtualBox and the project can use it.
    - Increment the score if the host has a GPU that the project can use.
    - Add the project's allocation balance.
    - Increment the score if the host is already attached to the project.
- Choose the 3 highest-scoring projects. If the host has a GPU that none of these can use, add the highest-scoring project that can use the GPU.
- If the user doesn't have an account on a selected project, initiate an account creation and tell the client to retry in 1 minute.
- If the client is currently attached to a project not in the top 3 but whose disk footprint exceeds 10 MB and whose score is nonzero, tell the client to remain attached with zero resource share.

The algorithm has lots of undetermined parameters. We'll guess appropriate values. An interesting research project would be to create a simulator for studying system behavior with different algorithms and parameters.

7. Project account creation

SU (transparently to the volunteer) creates accounts on various projects with random (anonymous) credentials. To minimize overhead, these accounts are created on demand.

SU creates project accounts using Web RPCs. These RPCs may take several seconds to complete, and they may temporarily fail (e.g. because the project is down). So we don't want to do them synchronously with user interactions. So we use the following approach:

- SU maintains a database table of project accounts, both established and pending. A periodic daemon process retries account creations that previously had transient failures.
- When a volunteer creates an SU account, SU identifies an initial set of projects, creates database records, and triggers the daemon. Typically the project accounts will be created within 10-20 seconds. Thus, by the time the volunteer installs and runs the BOINC client, the project accounts will exist and the client will be able to begin fetching jobs and computing immediately.
- When a client does an AM RPC to SU, the RPC handler identifies a set of projects the client should run (see below). For some of these, a project account may not already exist. The RPC handler creates database records, triggers the daemon, and tells the client to repeat the request in 60 seconds.

8. Implementation notes

We are using existing BOINC web code (PHP, database schema) as a basis for implementing SU. The BOINC web code (used for project web sites) has many features needed by SU:

- Account creation and login.
- Computing preferences.
- List of hosts.
- Message boards, private messages, “friends”, and other community-oriented features.

The code is configurable so that features related to job processing (which are not relevant to SU) can be disabled.

We have added a number of web pages and tables that are specific to SU: projects, allocations, project and volunteer keywords, project accounts, accounting records, and so on.

We have implemented a separate web interface for SU admins. This interface supports adding and editing projects, viewing accounting graphs, and so on.